BINOMIAL HEAPS

Componență echipă:

Dragomir Elena Alexandra

Apostu Alexandru Mihai

Gurzu Antonio Marco

Facultatea de Matematică și Informatică

Anul II, Grupa 212

Test Timp de Rulare

(Binomial Heap)

Am generat următoarele teste in Python(test_generator.py) folosind funcția rand() pentru intervalele descrise mai jos și pentru numărul de valori pe care le vrem din interval, pentru testele 3-6 și range(), pentru primele 2 teste.

1. sortedNumbers
   Primele 2.000.000 de numere naturale, în ordine crescătoare
   $2.000.000_{(10)} = 111101000010010000000_{(2)}$

2. reverseSorted
   Primele 2.000.000 de numere naturale, în ordine descrescătoare

3. randomSmalData
   100 numere aleatoare în intervalul [-1000, 1000]
   $100_{(10)} = 1100100_{(2)}$

4. randomMediumData
   10.000 numere aleatoare în intervalul [-1.000.000.000, 1.000.000.000]
   $10.000_{(10)} = 10011100010000_{(2)}$

5. randomLargeData
   2.000.000 numere aleatoare în intervalul [INT_MIN, INT_MAX]

6. tree4096
   $4096 = 2^{12}$ numere aleatoare în intervalul [-1.000.000, 1.000.000]
   $4096_{(10)} = 1000000000000_{(2)}$

Nota: **TOATE** rezultatele sunt exprimate în **MICROSECUNDE.**

(1 microsecunda = 1e-6 secunde)

Test Inserare – O(n log n)

Nume fișier Nr. Test	Random Large Data	Random Medium Data	Random Small Data	Sorted Numbers	Reverse Sorted	Tree4096
1.	811865	3604	0	707143	724870	1996
2.	817407	4031	0	731283	736772	1998
3.	864723	4988	0	726062	738066	1993
4.	831654	3988	1001	741831	741348	2042
5.	832326	4983	0	718143	731078	2023
6.	801025	4013	0	714364	740430	2035
7.	880894	3996	0	719992	733550	2031
8.	827165	3994	998	725777	722921	1998
9.	833107	3986	0	722291	750608	1031
10.	817419	5035	0	719112	735390	1994
Media	831758.5	4261.8	199.9	722599.8	735503.3	1914.1

- valorile sunt pentru inserarea întregii structuri de date, adică pentru o complexitate de O(n log n)
- observam că timpii depind cel mai mult de numărul de date introduse și obținem valori asemănătoare indiferent de ordinea numerelor din fișier

Test Ștergere - O(log n)

Nume fișier Nr. Test	Random Large Data	Random Medium Data	Random Small Data	Sorted Numbers	Reverse Sorted	Tree4096
1.	25926	0	0	27898	0	0
2.	16994	0	0	8037	8014	0
3.	19952	944	0	17951	25914	0
4.	22942	0	0	52371	16956	1000
5.	8978	0	0	46877	44956	0
6.	47937	1042	0	15955	3026	0
7.	40959	0	0	37296	26169	0
8.	16023	744	0	27294	34417	0
9.	32912	0	0	43888	27927	0
10.	23936	0	0	1001	25969	0
Media	25655.9	273	0	27856.8	21334.8	100

- asemănător cu inserarea, și ștergerea depinde cel mai mult de numărul de date introduse

Test Este_in – O(log n)

Nume fișier Nr. Test	Random Large Data	Random Medium Data	Random Small Data	Sorted Numbers	Reverse Sorted	Tree4096
1.	14029	1995	2995	15931	997	5983
2.	22938	2177	1992	28912	6724	2992
3.	26929	1994	2991	19947	14955	1998
4.	17757	1033	2440	22811	8973	2608
5.	6981	2997	2990	15954	27983	2965
6.	12968	1998	1994	16955	23941	2989
7.	16956	2770	2996	12969	24699	1994
8.	24932	1984	2478	8977	21943	2702
9.	2991	2463	1002	5034	14145	2993
10.	3995	2080	1920	2994	15879	1984
Media	15047.6	2149.1	2379.8	15048.4	16023.9	2920.8

Test Minim - O(log n)

Nume fișier Nr. Test	Random Large Data	Random Medium Data	Random Small Data	Sorted Numbers	Reverse Sorted	Tree4096
1.	1219	5983	4985	995	4867	5987
2.	4986	3996	4026	4017	1993	7216
3.	4912	4733	4989	3988	4988	3810
4.	4033	2990	3724	4986	3128	4989
5.	997	4983	2991	5908	3991	8015
6.	2992	6049	4025	3990	1994	9670
7.	4925	4986	3992	5622	3988	3991
8.	7979	2995	8255	3989	3613	3951
9.	2991	3994	3000	3994	5981	3987
10.	1997	3990	5989	3993	3986	4052
Media	3703.1	4469.9	4597.6	4148.2	3852.9	5566.8

- având o complexitate O(log n), numărul de operații făcute în această funcție nu variază mult cu creșterea numărului de date introduse. Numărul de rădăcini prin care se va căuta minimul din testele date este 7, 5, 3, 7, 7, respectiv 1

Test Maxim - O(n)

Nume fișier Nr. Test	Random Large Data	Random Medium Data	Random Small Data	Sorted Numbers	Reverse Sorted	Tree4096
1.	83743	2716	3147	67274	68917	4011
2.	79700	4987	7841	78771	66139	5984
3.	76800	5985	4013	65815	64885	5945
4.	79300	6763	4735	94785	67229	5018
5.	72720	3986	4308	88762	65885	5906
6.	77759	4099	3992	72885	61870	4990
7.	82770	5986	4533	75586	61907	4016
8.	77343	3019	2987	69633	65853	2996
9.	75929	5041	4989	69877	66780	3992
10.	71353	4990	2992	70878	62831	9963
Media	77741.7	4757.2	4533.7	75426.6	65229.6	5282.1

- Fiind o complexitate O(n) se observă că timpul de execuție crește liniar cu valoarea lui n

Test Succesor – O(n)

Nume fișier Nr. Test	Random Large Data	Random Medium Data	Random Small Data	Sorted Numbers	Reverse Sorted	Tree4096
1.	129592	2990	2922	132842	96792	2995
2.	119865	4988	2993	126569	103757	4691
3.	139896	2999	4818	124535	105697	2968
4.	127626	3989	3833	120753	106492	3101
5.	131675	996	3984	122100	119747	3993
6.	133672	4041	2993	111435	120574	4945
7.	127658	4060	3990	123674	112502	2826
8.	151906	2991	4985	105751	111565	2992
9.	138298	4980	4727	101988	110676	2995
10.	132079	4010	2991	39897	50863	2823
Media	133226.7	3604.4	3823.6	110954.4	103866.5	3432.9

Test Predecesor – O(n)

Nume fișier Nr. Test	Random Large Data	Random Medium Data	Random Small Data	Sorted Numbers	Reverse Sorted	Tree4096
1.	42526	2160	3511	134930	90529	2992
2.	121717	1998	3992	132084	99962	3123
3.	128667	4350	2992	127718	110536	1758
4.	129653	3026	2038	113731	101853	2995
5.	117195	4664	1994	114029	125185	3999
6.	130685	4697	3001	111683	121675	4046
7.	142401	2991	2289	121673	118802	2991
8.	140590	3808	2116	114693	113758	3654
9.	112549	2898	3335	106333	111701	3991
10.	120730	3710	3017	109216	109501	4939
Media	118673.3	3430.2	2828.5	118609	110350.2	3448.8

- Atât succesorul cât și predecesorul au, de asemenea, complexitate O(n), deci timp liniar de execuție

Test Cardinal – O(log n)

Nume fișier Nr. Test	Random Large Data	Random Medium Data	Random Small Data	Sorted Numbers	Reverse Sorted	Tree4096
1.	1930	5026	9375	4990	9782	6202
2.	3997	6367	4640	2029	1992	5964
3.	1996	4951	5500	6374	5991	5513
4.	4982	2740	5985	3993	3976	4951
5.	2961	4115	4986	4986	5689	1992
6.	4074	5543	3788	3879	4990	6012
7.	2992	2963	1994	1722	1997	2993
8.	4000	4986	5988	4948	2519	4615
9.	5339	4966	3989	2995	1992	5007
10.	2956	4752	3565	5829	2995	4733
Media	3522.7	4640.9	4981	4174.5	4192.3	4798.2

- Timpii sunt aproximativ egali cu cei de la minim deoarece se parcurge aceeași listă de rădăcini

Test K-element – O(n log n)

Nume fișier Nr. Test	Random Large Data	Random Medium Data	Random Small Data	Sorted Numbers	Reverse Sorted	Tree4096
1.	649032	7732	4106	619001	674234	4987
2.	1487693	8004	3551	728457	736597	3810
3.	2331557	9993	2512	806894	807165	6902
4.	3178382	10972	2954	906680	894875	3978
5.	3994581	11966	5227	991992	980314	6981
6.	4007479	13002	1997	1006046	985531	7151
7.	4891433	15985	1986	1083938	1049470	3990
8.	5661983	16246	5037	1156474	1129061	4984
9.	6389808	15984	4992	1259912	1208161	6337
10.	6922078	20944	1939	1332392	1300880	8974
Media	3951403	13082.8	3430.1	989178.6	976628.8	5809.4

- Cum log n este aproximativ egal în testele date, timpul crește liniar cu numărul de date introduse
- La testele cu valori sortate obținem timpi de aproximativ 3 ori mai mici decât la cel cu Random Large Data, deși toate 3 au 2.000.000 de numere introduse

In tabelul alăturat se poate observa rata de creștere a funcțiilor bazat pe complexitatea lor:

	constant	*logarithmic*	*linear*	*N-log-N*	*quadratic*	*cubic*	*exponential*
n	**O(1)**	**O(log *n*)**	**O(*n*)**	**O(*n* log *n*)**	**O(n^2)**	**O(n^3)**	**O(2^n)**
1	1	1	1	1	1	1	2
2	1	1	2	2	4	8	4
4	1	2	4	8	16	64	16
8	1	3	8	24	64	512	256
16	1	4	16	64	256	4,096	65536
32	1	5	32	160	1,024	32,768	4,294,967,296
64	1	6	64	384	4,069	262,144	1.84×10^{19}

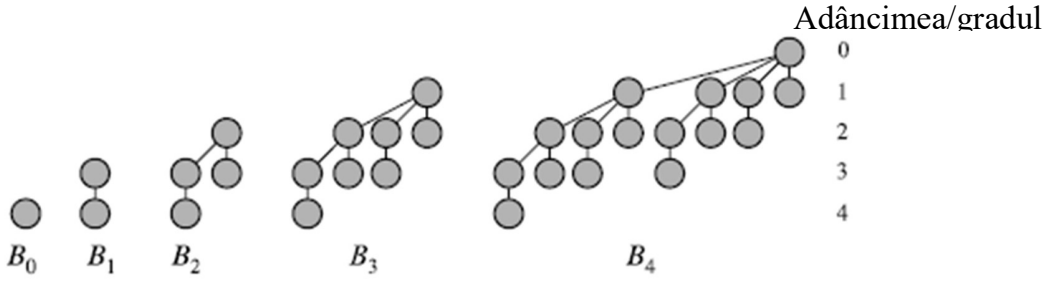
Complexitățile operațiilor din Binomial Heap:

Funcție	Sintaxă	Descriere	Complexitate
1. Inserare	insereaza(H ,x)	Inserează elementul x în mulțimea H	O(log n)
2. Ștergere	sterge(H, x)	Șterge elementul x din mulțimea H	O(log n)
3. Min	findMin(H)	Returnează elementul minim din mulțimea H	O(log n)
4. Max	findMax(H, INT_MIN)	Returnează elementul maxim din mulțimea H	O(n)
5. Succesor	successor(H, x)	Returnează succesorul elementului x din mulțimea H (dacă există)	O(n)
6. Predecesor	predecessor(H, x)	Returnează predecesorul elementului x din mulțimea H (dacă există)	O(n)
7. Al k-lea element	k_element(H, k)	Returnează al k-lea element din mulțimea H în ordine crescătoare	O(n log n)
8. Cardinalul	cardinal(H)	Returnează numărul de elemente din H, dar fără să numere elementele	O(log n)
9. Căutare nod	este_in(H, k)	returnează nodul cu valoarea k dacă x ∈ H și NULL altfel.	O(log n)

Motivație:

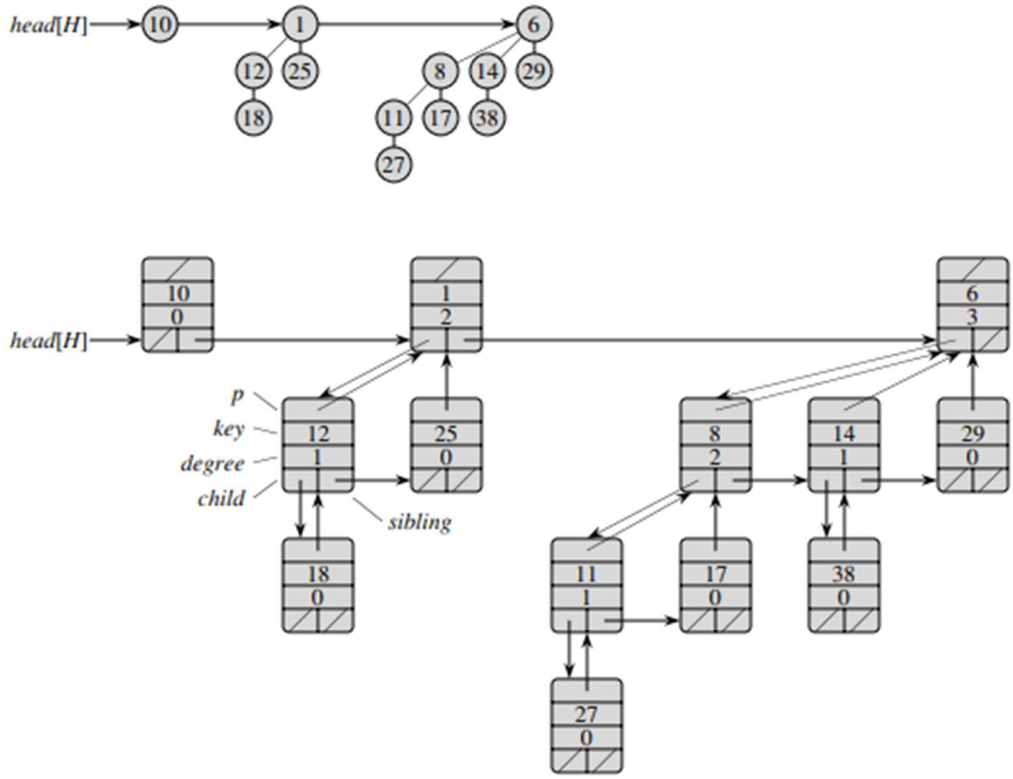
Heap-ul binomial este o structură de date care se comportă ca o coadă cu prioritate dar, spre deosebire de Binary Heap, suportă mai eficient operația de Merge si Union a doua heap-uri.

Heap-ul binomial este format dintr-o mulțime de arbori binomiali unde fiecare arbore binomial satisface proprietatea de ordonare a unui heap: cheia unui nod este mai mare sau egală decât cheia părintelui său și există cel mult un arbore binomial a cărui rădăcină are un grad dat. Arborele binomial se construiește recursiv, astfel un arbore de grad k este format din 2 arbori de grad k-1, unul din ei fiind atașat ca fiu al rădăcinii celuilalt în funcție de ordinea celor 2 rădăcini pentru a se păstra regula ca rădăcina arborelui sa aibă valoarea cea mai mică. Astfel, un arbore binomial de ordin k are 2^k noduri și este reprezentat astfel:



Pentru a construi un heap binomial cu n rădăcini vom avea o înlănțuire de arbori binomiali B_k, fiecare cu câte 2^k noduri. Având scriere binară unică a numărului n, vom putea pune câte un arbore de grad k la fiecare apariție a lui 1 din scrierea binară, obținând deci o scriere a lui n ca sumă de puteri ale lui 2, identificând astfel structura arborilor conținuți în heap. De exemplu, pentru primul test avem n = 2.000.000 care are scrierea binară 11110100000100100000000. Vom avea deci 7 arbori binomiali cu gradele 7, 10, 15, 17, 18, 19, respectiv 20.

Structura unui heap binomial:



Pentru fiecare nod se reține valoarea sa, gradul și pointer către “parent”, „child” și “sibling”. Întrucât numărul de “copii” avuți de fiecare nod diferă pentru fiecare grad, se va reține doar pointer către nodul stâng, “child”, iar pentru a accesa celelalte noduri de pe același nivel, se va folosi pointer către nodul drept al fiecăruia, “sibling”, ca într-o listă simplu înlănțuită.

Avantaje și dezavantaje:

Heap-urile binomiale sunt o extensie a heap-urile binare care oferă o complexitate mai mică operațiilor de Union sau Merge și suportă toate funcțiile heap-urilor binare într-un timp eficient.

În Binary Heap, heap-ul este format dintr-un singur arbore, care este, de fapt, un arbore binar, în timp ce în cazul heap-urilor binomiale, acestea sunt formate dintr-o colecție de arbori binomiali înlănțuiți.

Avantaje:

- Operația de Merge sau Union a doua heap-uri binomiale în complexitate O(log n)

Se înlănțuie arborii binomiali din cele doua heap-uri în ordine crescătoare după grad, se verifică dacă există arbori cu același grad k și în caz afirmativ se unesc arborii respectivi într-un arbore de grad k-1. Se continuă procedeul până când toți arborii au grade diferite.

- Operația de inserare, ștergere și găsire a minimului în O(log n)

Pentru inserare se construiește un heap format doar din nodul de adăugat și se face Union între acest heap si cel inițial. Pentru găsirea minimului se parcurg doar rădăcinile arborilor din heap in O(log n). Pentru ștergere se aduce nodul de șters până la rădăcina arborelui scăzându-i cheia până la un minim absolut (INT_MIN) și se extrage acest minim.

Dezavantaje:

- Timpul de parcurgere pentru găsirea unui element în O(n)
- Nu există o ordine crescătoare a elementelor
- Pentru al k-lea element trebuie sa facem o copie în O(n log n), iar din copie extragem nodul minim de k ori, adică O(k log n)

Sales pitch:

Heap-ul binomial este structura de date perfectă pentru operații simple precum inserarea, ștergerea și extragerea elementelor însă punctul forte al acestei structuri care îl diferențiază de altele este operația de Union a două heap-uri în cel mai scurt timp (O(log n)). Această operație ajută mult când avem nevoie sa unificăm mai multe seturi de date cât mai repede.

Link GitHub: https://github.com/ApostuAlexandru/BinomialHeap/